

15장 let, const 키워드와 블록 레벨 스코프

15.1 var 키워드로 선언한 변수의 문제점

1. 변수 중복 선언 허용

```
var x = 1;
var y = 1;

// var 키워드로 선언된 변수는 같은 스코프 내에서 중복 선언을 허용한다.
// 초기화문이 있는 변수 선언문은 자바스크립트 엔진에 의해 var 키워드가 없는
var x = 100;
// 초기화문이 없는 변수 선언문은 무시된다.
var y;

console.log(x); // 100
console.log(y); // 1
```

2. 함수 레벨 스코프

함수 외부에서 var 키워드로 선언한 변수는 코드 블록 내에서 선언해도 모두 전역 변수가 된다.

```
var x = 1;

if(true) {
  // x는 전역 변수다. 이미 선언된 전역 변수 x가 있으므로 x 변수는 중복
  // 이는 의도치 않게 변수값이 변경되는 부작용을 발생시킨다.
  var x = 10;
}

console.log(x); // 10
```

3. 변수 호이스팅

변수 호이스팅에 의해 var 키워드로 선언된 변수는 변수 선언문 이전에 참조할 수 있다.

```
// 이 시점에는 변수 호이스팅에 의해 이미 foo 변수가 선언되었다(1. 선언 단계)
// 변수 foo는 undefined로 초기화된다.(2. 초기화 단계)
console.log(foo); // undefined

// 변수에 값을 할당(3. 할당 단계)
foo = 123;

console.log(foo); // 123

// 변수 선언은 런타임 이전에 자바스크립트 엔진에 의해 암묵적으로 실행된다.
var foo;
```

15.2 let 키워드

1. 변수 중복선언 금지

```
var foo = 123;
// var 키워드로 선언된 변수는 같은 스코프 내에서 중복 선언을 허용한다.
// 아래 변수 선언문은 자바스크립트 엔진에 의해 var 키워드가 없는 것처럼 동작

var foo = 456;

let var = 123;
// let이나 const 키워드로 선언된 변수는 같은 스코프 내에서 중복 선언을 허용하지 않는다.
let var = 456; // SyntaxError: Identifier 'var' has already been declared
```

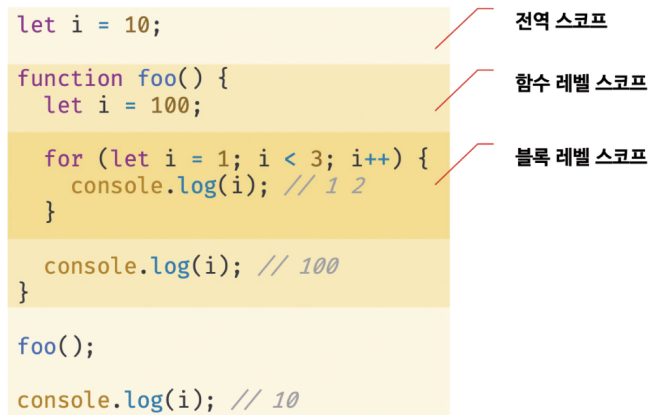
2. 블록 레벨 스코프

let 키워드로 선언한 변수는 모든 코드 블록(함수, if 문, for 문, while 문, try/catch 문 등)을 지역 스코프로 인정한다.

```
let foo = 2; // 전역 변수

{ let foo = 2; // 지역 변수
  let var = 3; // 지역 변수
}

console.log(foo); // 1
console.log(bar); // ReferenceError: bar is not defined
```



The diagram illustrates three levels of variable scope in JavaScript:

- 전역 스코프 (Global Scope):** Represented by the line `let i = 10;` at the top.
- 함수 레벨 스코프 (Function Level Scope):** Represented by the block `function foo() { let i = 100; ... }`.
- 블록 레벨 스코프 (Block Level Scope):** Represented by the `for` loop block `for (let i = 1; i < 3; i++) { console.log(i); }`.

Additional code shown includes `console.log(i); // 100` (accessing the function-level `i`) and `foo(); console.log(i); // 10` (accessing the global `i`).

<https://velog.io/@kozel/모던-자바스크립트-15장-let-const-키워드와-블록-레벨-스코프>

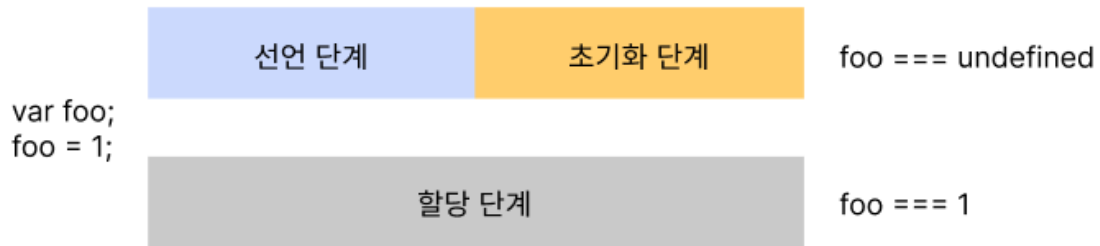
3. 변수 호이스팅

- var 키워드

```
// var 키워드로 선언한 변수는 런타임 이전에 선언 단계와 초기화 단계가 실행
// 따라서 변수 선언문 이전에 변수를 참조할 수 있다.
console.log(foo); // undefined

var foo;
console.log(foo); // undefined

foo = 1; // 할당문에서 할당 단계가 실행된다.
console.log(foo); // 1
```



var 변수의 생명주기

<https://velog.io/@koseony/javascript-뿌시기-var-let-const의-특징과-차이점>

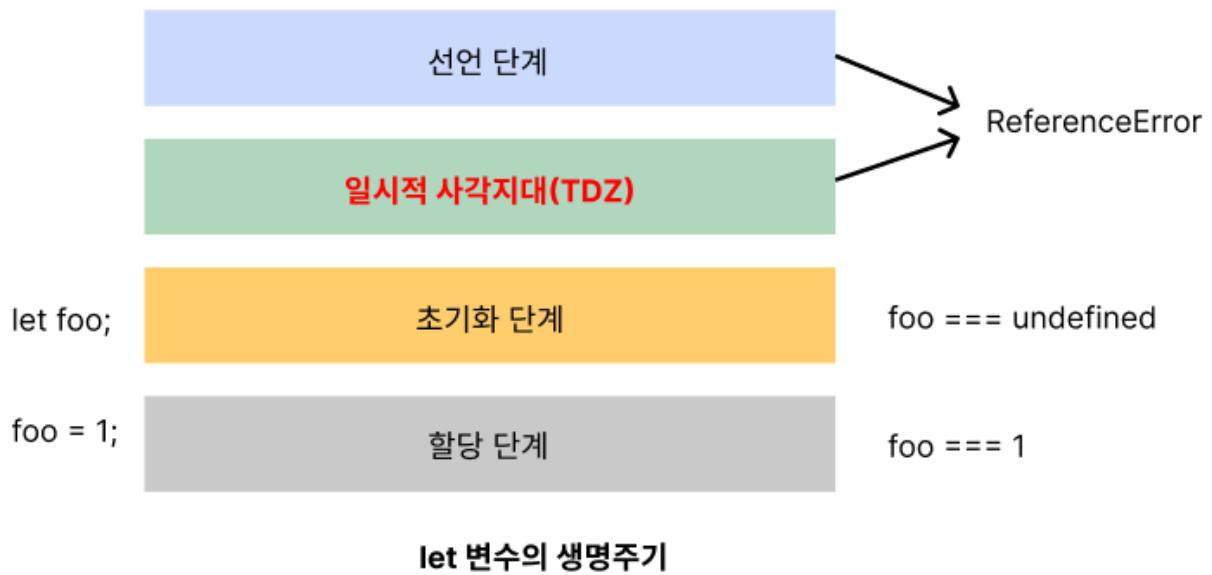
- let 키워드

let 키워드로 선언한 변수는 "선언 단계"와 "초기화 단계"가 분리되어 진행된다.

```
// 런타임 이전에 선언 단계가 실행된다. 아직 변수가 초기화되지 않았다.
// 초기화 이전의 일시적 사각지대에서는 변수를 참조할 수 없다.
console.log(foo); // ReferenceError: foo is not defined

let foo; // 변수 선언문 단계에서 초기화 단계가 실행된다.
console.log(foo); // undefined

foo = 1; // 할당문에서 할당 단계가 실행된다.
console.log(foo); // 1
```



<https://velog.io/@koseony/javascript-뿌시기-var-let-const의-특징과-차이점>



그렇다면 `let` 키워드는 변수 호이스팅이 발생하지 않는걸까?

```
let foo = 1; // 전역 변수

{
  console.log(foo); // ReferenceError: Cannot access 'foo'
  let foo = 2; // 지역 변수
}
```



변수 호이스팅이 없다면 전역 변수 `foo`의 값을 출력해야 하나, 여전히 호이스팅이 발생하기에 참조 에러가 발생한다.

4. 전역 객체와 `let`

`var` 키워드로 선언한 전역 변수와 전역 함수, 그리고 선언하지 않은 변수에 값을 할당한 암묵적 전역은 전역 객체 `window`의 프로퍼티가 된다.

```

// 이 예제를 브라우저에서 실행해야 한다.

// 전역 변수
var x = 1;
// 암묵적 전역
y = 2;
// 전역 함수
function foo() {}

// var 키워드로 선언한 전역 변수는 전역 객체 window의 프로퍼티다.
console.log(window.x); // 1
// 전역 객체 window의 프로퍼티는 전역 변수처럼 사용할 수 있다.
console.log(x); // 1

// 암묵적 전역은 전역 객체 window의 프로퍼티다.
console.log(window.y); // 2
console.log(y); // 2

// 함수 선언문으로 정의한 전역 함수는 전역 객체 window의 프로퍼티다.
console.log(window.foo); // function foo() {}
// 전역 객체 window의 프로퍼티는 전역 변수처럼 사용할 수 있다.
console.log(foo); // function foo() {}

```

15.3 const 키워드

- const 키워드는 상수를 선언하기 위해 사용한다.

1. 선언과 초기화

- const 키워드로 선언한 변수는 반드시 선언과 동시에 초기화해야 한다.

```
const foo = 1
```

- let 키워드와 마찬가지로 블록 레벨 스코프를 가지며, 변수 호이스팅이 발생하지 않는 것처럼 동작한다.

2. 재할당 금지

- `const` 키워드로 선언한 변수는 재할당이 금지된다.

```
const foo = 1;
foo = 2; //TypeError: Assignment to constant variable.
```

3. 상수

- 상수는 재할당이 금지된 변수이다.
- 상태 유지와 가독성, 유지보수의 편의를 위해 적극 사용해야 한다.
- 일반적으로 상수의 이름은 대문자로 선언해 상수임을 명확히 나타낸다.
 - 여러 단어로 이뤄진 경우에는 언더스코어(_)로 구분해서 스네이크 케이스로 표현하는 것이 일반적이다.

4. `const` 키워드와 객체

- `const` 키워드로 선언된 변수에 객체를 할당한 경우 값을 변경할 수 있다.
 - 재할당 없이도 직접 변경이 가능하기 때문이다.

```
const person = {
  name: 'kwak'
}

// 객체는 변경 가능한 값이다. 따라서 재할당 없이 변경이 가능하다.
person.name = 'lee';

console.log(person); // {name: 'lee'}
```

15.4 `var` vs. `let` vs. `const`

- ES6를 사용한다면 `var` 키워드는 사용하지 않는다.

- 재할당이 필요한 경우에 한정해 `let` 키워드를 사용한다.
 - 이때 변수의 스코프는 최대한 좁게 만든다.
- 변경이 발생하지 않고 읽기 전용으로 사용하는(재할당이 필요 없는 상수) 원시 값과 객체에는 `const` 키워드를 사용한다.
 - `const` 키워드는 재할당을 금지하므로 `var`, `let` 키워드보다 안전하다.